**Software Design**

**Document**

**for**

**QTC PDF Web Viewer**

**Version 1.0 approved**

**Prepared by Cristian Corrales, Chandel Buelna, Phoebe Castanedo, Brandon Gonzalez, David Sanchez, Joaquin Robles, Sean Ybarra Roman Arias, and Ares Ton-That**

**Sponsored by QTC**

**12/5/2020**

Table of Contents................................................................................................................... pg 2

Revision History..................................................................................................................... pg 4

1. Introduction................................................................................................................ pg 5
   1. Purpose........................................................................................................... pg 5
   2. Document Conventions……………………................................................. pg 5
   3. Intended Audience and Reading Suggestions................................................ pg 5
   4. System Overview........................................................................................... pg 5
2. Design Considerations............................................................................................... pg 6
   1. Assumptions and dependencies..................................................................... pg 6
   2. General Constraints........................................................................................ pg 6
   3. Goals and Guidelines...................................................................................... pg 6
   4. Development Methods................................................................................... pg 6
3. Architectural Strategies.............................................................................................. pg 7
4. System Architecture................................................................................................... pg 8
   1. ....................................................................................................................... pg 9
   2. ....................................................................................................................... pg 10
5. Policies and Tactics.................................................................................................... pg 12
   1. Specific Products Used.................................................................................. pg 12
   2. Requirements traceability............................................................................... pg 12
   3. Testing the software....................................................................................... pg 12
   4. Engineering trade-offs................................................................................... pg 13
   5. Guidelines and conventions........................................................................... pg 14
   6. Protocols........................................................................................................ pg 14
   7. Maintaining the software............................................................................... pg 14
   8. Interfaces........................................................................................................ pg 14
   9. System's deliverables..................................................................................... pg 14
   10. Abstraction..................................................................................................... pg 14
6. Detailed System Design............................................................................................. pg 15

6.1 UI/UX............................................................................................................ pg 15

6.1.1 Responsibilities...................................................................................... pg 15

6.1.2 Constraints............................................................................................. pg 15

6.1.3 Composition........................................................................................... pg 15

6.1.4 Uses/Interactions.................................................................................... pg 16

6.1.5 Resources............................................................................................... pg 16

6.1.6 Interface/Exports.................................................................................... pg 17

6.2 Server Side Logic........................................................................................... pg 17

6.2.1 Responsibilities...................................................................................... pg 17

6.2.2 Constraints............................................................................................. pg 17

6.2.3 Composition........................................................................................... pg 17

6.2.4 Uses/Interactions.................................................................................... pg 18

6.2.5 Resources............................................................................................... pg 18

6.2.6 Interface/Exports.................................................................................... pg 18

6.3 PDF Database................................................................................................ pg 18

6.3.1 Responsibilities...................................................................................... pg 18

6.3.2 Constraints............................................................................................. pg 19

6.3.3 Composition........................................................................................... pg 19

6.3.4 Uses/Interactions.................................................................................... pg 19

6.3.5 Resources............................................................................................... pg 20

6.3.6 Interface/Exports.................................................................................... pg 20

1. Detailed Lower level Component Design................................................................... pg 21

7.1 Index.cshtml.................................................................................................... pg 21

7.1.1 Classification.......................................................................................... pg 21

7.1.2 Processing Narrative(PSPEC)................................................................ pg 21

7.1.3 Interface Description.............................................................................. pg 21

7.1.4 Processing Detail.................................................................................... pg 21

7.1.4.1 Design Class Hierarchy................................................................... pg 21

7.1.4.2 Restrictions/Limitations.................................................................. pg 21

7.1.4.3 Performance Issues........................................................................ pg 21

7.1.4.4 Design Constraints......................................................................... pg 21

7.1.4.5 Processing Detail For Each Operation........................................... pg 21

7.2 DocViewerHandler.cshtml.............................................................................. pg 21

7.2.1 Classification.......................................................................................... pg 21

7.2.2 Processing Narrative(PSPEC)................................................................ pg 22

7.2.3 Interface Description.............................................................................. pg 22

7.2.4 Processing Detail.................................................................................... pg 22

7.2.4.1 Design Class Hierarchy................................................................... pg 22

7.2.4.2 Restrictions/Limitations.................................................................. pg 22

7.2.4.3 Performance Issues........................................................................ pg 22

7.2.4.4 Design Constraints......................................................................... pg 22

7.2.4.5 Processing Detail For Each Operation........................................... pg 22

1. Database Design......................................................................................................... pg 23
2. User Interface............................................................................................................. pg 24
3. 9.1 Overview of User Interface.......................................................................... pg 24
4. 9.2 Screen Frameworks or Images..................................................................... pg 24
5. 9.3 User Interface Flow Model........................................................................... pg 25
6. Requirements Validation and Verification................................................................. pg 26
7. Glossary..................................................................................................................... pg 27
8. References................................................................................................................. pg 27

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Date | Reason For Changes | Version |
| Initial Draft | 11/20/2020 | This is the initial draft of the document created. |  |
| Requirements | 12/26/2020 | Revision of initial requirements based on feedback and additions from QTC. |  |
| Final Draft | 4/18/20 | Revisions of overall System Introduction, upper level system design, lower level system design, user interface, and requirements validation |  |
|  |  |  |  |

**1. Introduction**

**1.1 Purpose**

The QTC PDF Web Viewer, which is currently in version 1.0 development, is a software tool designed to give users the ability to view multiple PDF files within the same viewer in a single browser. This application gives the user the ability to scroll past one document on to the next, allowing them to look through multiple files with ease. Additionally, the application lets the user search keywords across all files in order to ease the process when looking for specific data. Finally, the application incorporates a filter mechanism that allows the user to select a category which will minimize the list of PDFs the user has access to in order to once again give the user more control of what data is being viewed. This software tool is currently a standalone software. QTC may build upon the software in the future, but no additional developments are planned in the near future.

**1.2 Document Conventions**

All sections and their consequent subsections will be bolded to improve ease of access of information contained within these sections. Descriptions and summaries will not have any bold or italicized text unless an important point must be highlighted with the document. Every requirement will have its own priority unless the team states otherwise.

**1.3 Intended Audience and Reading Suggestions**

Managers should read the system overview in the section below and the requirements section in section 10. Marketing staff need to only read the system overview. Developers and testers should read the entire document to gain a better understanding of the system architecture and design as well as its high-level requirements. Documentation writers will find much of the document useful as well.

This design document contains a system overview, design constraints, software development tactics used during the course of product development, system architecture, system design, database design, user interface descriptions, high-level requirements specifications, a glossary to improve every reader’s understanding of the document, and a list of all available references.

**1.4 System Overview**

The PDF Web Viewer is designed to improve QTC’s efficiency when accessing sensitive PDF files. The viewer allows several PDFs to be viewed continuously through one web page. By providing a primary search field, the viewer allows users to access all PDFs pertaining to a specific keyword. All PDFs will then be brought into view and can be accessed simply by scrolling through each page within the viewer. A secondary search field allows users to access specific information by entering a keyword or phrase that can be found somewhere within the documents.

**2. Design Considerations**

**2.1 Assumptions and Dependencies**

Since this is a web-based program, an internet connection is required to use the program. The Web Viewer should work on any operating system as it is a web-based viewer. The data may be updated frequently as new PDFs are added to the database. Users should be familiar with how the metadata of PDFs are assigned to make finding relevant results easier.

**2.2 General Constraints**

Since this is a product for QTC, any specifications they require during development must be included. The size and security of the database used must be appropriate for extensive use of potentially sensitive data. Since the organization of the data is heavily reliant on metadata, how the metadata is assigned to PDFs may influence the structure of the program.

**2.3 Goals and Guidelines**

The delivery date for this software is the end of Spring 2021.

**2.4 Development Methods**

We are following the Agile Development model to work on the project, as it is QTC’s method as well. There are weekly sprints in which a goal is to be met followed by a meeting with QTC with updates regarding the goal and any future goals for the next week.

**3. Architectural Strategies**

The team uses a handful of particular types of products. C#, ASP.NET, Microsoft SQL Server Management Studio (abbreviated as SSMS), and Atalasoft’s DotImage SDK are the most influential products we use that come to mind. C#, ASP.NET, and SSMS in specific are used for the project because they are designed by Microsoft to be integrated to work together among one another. This makes web development intuitive and flexible considering we will have the software written in C# communicating with the SQL server.

Atalasoft’s DotImage SDK comes equipped with an embedded PDF viewer that has the function of displaying pages continuously in a single scrollable form, and is the prominent form of existing software components we will reuse. It serves as the basis for fulfilling the software requirement wherein the software shall display the PDFs on the page browser. We will later add the functionality for the user to scroll past one PDF file onto the next.

MVC is the software architecture model we are using as instructed by QTC, and it stands for “Model View Controller”. Model is the program’s data model, View is what the frontend or what the user sees, and Controller manipulates the model and updates the view. As for memory management policies we are concerned primarily with storing information in the database pertaining to the PDFs. In terms of data storage and management, our team utilizes MySQL to maintain data correlating to PDFs, and the software will additionally function with local files. Our team’s future plans mainly include a feature that allows the user to update a PDF with new attributes which in turn will update correspondingly in the MySQL database.

**4. System Architecture**

This system starts with our users, they will interact with our PDF web application through some basic inputs. Those inputs are then processed through the web application and then served as queries to the PDF database where the PDF files’ information, such as the local address of the file, is housed at. The queries then retrieve either a singular file location or a list of file locations and serve them back to the application where they are processed and normalized for human viewing. The whole system finishes with the user being able to view the desired PDF files, or they can repeat the process with new inputs that create new queries.



Our users are the medical practitioners who have access to QTC’s web application for the benefit of storing and retrieving pdf files associated with their practice. The users main responsibility of the system is to provide desired inputs in order to operate the web application for a desired outcome regarding the stored PDF documents. They can view and search documents according to their inputs and the ability to do all that comes from the inner workings of the integrated Atalasoft SDK.

The major responsibilities of the PDF web application can be split up according to the MVC architecture. All the functionality of the front end/UI/UX is primarily housed in the view files of the project. These are responsible for the user interface which is how users will interact with the app and where the pdf documents will be accessible for viewing. The controllers are the server side portion of the project where the filtering methods are implemented. The client side, which consists of JavaScript, will add the functionality which is tied to the viewer. That is, it will handle the page scrolling as well and the searching of keywords within the PDF files. This is more of the analytical side of the project and where all the inner workings are abstracted from the user.

The database is the models portion of the project and this is where we store all the document’s information for the project. The database consists of a table that will columns for specific data pertaining to a specific PDF file. MySQL will be the foundation for the database.

As you can see the web application we are developing can be broken down even further into individual components. Each component will be broken down according to the MVC architecture so there will be a frontend, server side, a database, and a client side portion of the project. The client side includes the Atalasoft library methods as a module as well because those methods are responsible for a lot of the functionality of the project and it serves us to be more detail orientated about the design diagram here. The frontend will be responsible for serving the user interface to the users so they can interact with the application, the server side will be doing all the abstract functions of the process that the user doesn’t need to concern themselves with knowing, and the Atalasoft libraries will be providing a lot of the main functionality of the process like the PDF viewer.

Module 4.1.1-Frontend: The frontend module will consist of all the html/view files for our project. The .cshtml file consists of mostly html and css and can work in conjunction with javascript and c#. They could also be aspx files of some sort depending on the design and the needs of our clients. The files will be responsible for displaying the user interface so that the user can operate the application. The user interface takes this interaction as input and sends it off to the server/client side portion of the application depending on the input.

Module 4.1.2 Server Side: The server side module is where all the logic happens with the application regarding the filtering mechanism. It is also an intermediary between the frontend and the database by receiving inputs that get turned into queries and processes them to the correct destination. It consists of the controller, and handler files.

Module 4.1.3-Client Side: This module is what makes all of the functionality of viewing PDF files possible. It is a library that consists of javascript, jquery, and css files. It works hand in hand with the frontend interface, but it also has some parallels with the server side as well. The client side also handles the logic for the search functionality with the system.



DFD Level 1

As you can see the frontend really consists of the user interface we will be displaying to the user to collect some desirable inputs. These inputs are then sent to the server side or client side portion of the application where they are filtered and processed depending on what they are. The filter mechanism will be handled by the server side. The type of query or call is completely dependent on what kind of inputs the user supplies from the frontend. The query could be a fetch to the database and depending on this query it determines what file information is forwarded back to the user. The controllers receive the forwarded file information and, through the client side, methods are then served to the users via a pdf viewer that is part of the javascript files in the library.

This design was determined by our company liaison for QTC because they were using MVC technologies in correspondence with c# in their company already, the Atalasoft library was also a familiar technology for them and they proposed we use it in the application as well. We looked in to substitute modules for the Atalasoft library because we were not sure if we would be able to gain a license for the SDK, but all those options were discarded once we found out we would be able to get a trial license.



Control Flow Diagram

**5. Policies and Tactics**

**5.1 Choice of which specific products used**

Atalasoft SDK, Visual Studio, Microsoft SQL Server Management Studio

**5.2 Plans for ensuring requirements traceability**

The team will be using the Agile methodology and the Jira Tool. The team will follow a scrum development strategy in weekly long sprints. The team will gather with the sponsors, QTC, to collaborate in the creation of user stories also known as project requirements. The user stories will contain the following: The description, which are essentially the project requirements. The date it was created, the date it was completed. The version of the application it belongs to. Sub-user stories if any, and lastly, the assignee. Furthermore, the Jira tool allows the team to place the user stories in a virtual whiteboard to track the progress of said user stories. Lastly, the Jira tool will keep track of all user stories created in what is called a “Backlog”, ensuring the traceability of all user stories, and therefore, requirements.

**5.3 Plans for testing the software**

The team will follow a test driven development strategy. This means that all the software requirements will be converted to test cases before the code / software feature is developed. After that, the software that has been created will be continuously tested against all test cases to ensure fully functioning code at all times. The test driven development cycle is as follows:

* Add a test:
  + Before adding a feature we must first add a test case. If the engineer is able to write a test case, that means that the engineer is fully aware of the feature requirements and they understand what is expected of the feature and/or new implementation proposed.
* Run all tests and see if the new test case fails:
  + If the new test case fails it assures the engineer of two things. One, the test case is looking for specific requirements, therefore, a failed test case lets the engineer know that the requirements have not been met by previous implementation. Two, a failed test case proves that it is not flawed and will only pass after the requirements are met.
* Write the code
  + Code the implementation that is required and have the goal of making that specific test pass
* Run all test cases
  + If the new test passes, this means we are clear to move onto the next step. If the test fails, we must carefully look over our implementation and adjust it until it meets the tests requirements
* Refactor Code
  + The first time the code passes does not imply that the code is perfect. Therefore, in this step, we look over at our passing implementation and refactor it. Here, we make sure that all variable names make sense, method names make sense, there is no duplication, and every method does at most 1 task. In the case when code needs to be split up, we may have to add test cases to the individual parts that are being split up.
* Repeat
  + If new features are proposed, we repeat the cycle and ensure that we have fully functioning code at all times.

**5.4 Coding Guidelines and Conventions**

**5.4.1 Source Control Standards**

The team will be using Git for local source control and Github as the online source control system. The senior design team lead will create a new project on their machine. The team lead will then proceed to create the appropriate folders for the UI they are

building (WPF, MVC, Angular, etc.) for the application. Once the majority of the

project is created with forms, pages, html, etc. and the lead is ready to start

assigning work to the rest of the team, he/she checks in the project to the source

control system, which is Github. Each team member may now retrieve the new project

from one location and it will automatically be created correctly on their computer.

**5.4.2 C# Programming Standards**

Classes, enum types, enum values, events, exception classes, read-only static fields, interfaces, methods, namespaces, properties, and public instance fields are Pascal Case. Local variables in a method, parameters to a method, and protected instance fields are Camel Case. Local variables (variables inside a method) should have a meaningful name that describe its usage and should be declared at the top of the method. Global variables (variables that can be accessed by any method) should begin their name with an underscore and be pascal case. Constant naming should be UPPER\_CASE with underscores. Class names should be nouns and should be singular e.g “Customer”. The plural form should be suffixed with “Collection”. Interface names should start with “I” e.g. “ICustomer”. Method names should be a verb as methods generally perform an action, e.g. “Delete” or “GetCustomers”. Methods should perform at most 1 action. Do not hard code numbers and or strings that will be presented to the user. Keep code out of the UI layer. Make code readable to avoid unnecessary comments but do use comments. Comments should contain: a summary of what the code does, if describing a method, define parameters and return types if any. Follow a test driven development strategy for any new features implemented. Use “string.Empty()” rather than two double quotes (“”). Use “string.IsNullorEmpty()” when checking for empty strings. Use the string builder class for creating large strings in code (Pre-init to a large size if needed). If and else statements, longside loops, should have open and close curly braces at all times.

**5.4.3 SQL Server Database Naming Standards**

The database name will be something that makes sense to the team and anyone else who is not familiar with the application. The table naming conventions are as follows: Table names should be singular, not plural. Primary keys for each table should either be an integer data type or GUID. Each table should have common columns that allow us to keep track of who is making updates to the table, column examples are “CreatedBy”, “CreatedDate”, “UpdatedBy”, and “UpdatedDate”. For large databases, we will break up each table into schemas. Column names should be pascal case and abbreviations should be avoided. Triggers should end with the suffix “Trigger”. User defined functions should have the suffix “Function”. Finally, all SQL statements and built in functions need to be upper case.

**5.5 Interfaces for end-users, software, hardware, and communications** I

We will be using the Atalasoft pdf viewer interface and adjust the components to our needs. More specifically we will be using the Atalasoft DotImage interface. The interface will implement the default components of the Atalasoft DotImage viewer. The team will then add extra user interface components such as a search bar for keyword searching and a filter for selecting specific categories. The filter, once clicked on, will show yet another user interface component. This component will list out the possible categories that the user may be able to filter by.

**6. Detailed System Design**

**6.1 UI/UX (Module)**

**6.1.1 Responsibilities**

The UI component of the QTC PDF Web Viewer is responsible for receiving the inputs from the user and redirecting those inputs to their corresponding module within the application. The UI is responsible for contacting the server side whenever the correct input from the user has been given. The UI is responsible for contacting the methods in the client side whenever the correct input from the user has been given. Lastly, the UI is responsible for providing html elements so that the client side may populate them, such as the PDF viewer.

The UI is responsible for collecting the specified filter categories by a user who wishes to filter the PDFs. A drop-down list containing all existing categories will be shown for this purpose. A text input will also be provided for users to enter keywords where it will automatically recommend categories in the drop-down list based on whatever partial string the user has input. It is the job of the UI to collect any categories that are selected therein and send this information to the server side where it will be processed.

The UI is responsible for collecting the specified search terms by a user who wishes to search across multiple PDF files and view the instances of the keyword being searched.

**6.1.2 Constraints**

The UI has no connection to the database, nor does it have access to Atalasoft SDK methods. The display queue (query list of PDF file paths) is populated by the server side logic module which receives the user's input into the UI that contains desired keywords. The only data storage/collection the UI is responsible for, is reading the paths to relevant files from the server side module, and receiving the desired categories from the user input.

Regarding assumptions made by this module; if there are no current filters constraining the access to the PDFs, the module will display the first PDF contained in the database and allow the user to scroll through and view all PDFs regardless of the content of their metadata. As of the current plans for this project, it is not possible to edit the PDF files using this viewer.

**6.1.3 Composition**

This module comprises 3 major components. The first contains the empty (at first) “div” html components that will later be used by the client side to create a PDF viewer and thumbnails. These components are empty at first. The client side requires empty div components to be used during the creation of the PDF viewer, this allows the PDF viewer to be displayed in the webpage after it has been created in the client side using javascript. Thus, the UI sets aside these components to be used later by the client side.

The second component of this module is the filter. This component is a simple drop-down menu with text input which allows the user to select their desired categories of PDFs, which are processed by the server side module. When properly processed, the UI will receive a new list of PDFs, which will be fed to the client side.

The last component of this module is the search bar. This component is a simple search bar that allows the user to input a text string that will be passed onto the client side to execute the corresponding logic for “search”.

**6.1.4 Uses/Interactions**

This UI entity has direct interaction with the user via their various inputs into the application. The scope of its interactivity with the user encompasses all possible requests that could be made by their input whether it is simple movement such as scrolling and zooming, or the pressing of buttons for the filter mechanism.

**6.1.5 Resources**

The UI module is not designed to handle any sort of data processing or logic outside of properly displaying its graphical components. To enable its interaction with the server side module, this module must be able to read the user input regarding which categories of PDFs shall be displayed, and store these category(s) once read.

Additionally, the UI/UX module, in of itself, of course requires some graphical resources, and the Javascript code embedded in its web pages requires the inclusion of the JQuery and Bootstrap library.

**6.1.6 Interface/Exports**

In order to start developing the frontend a UI/UX package was imported into the project with QTC’s color palettes and design standards. The package was obtained from QTC’s design team and with the purpose of facilitating the development of the frontend to comply with their standards. This UI/UX packaged served its purpose and tremendously helped with the total and successful completion of the front end to QTC’s desired standards. If future development or updates to the frontend occur this package will and could be updated as well to improve on the UI/UX design of the viewer.

**6.2 Server Side Logic (Module)**

**6.2.1 Responsibilities**

The server side module of this product is responsible for enabling the communication between user and database. Specifically, it must receive data from the UI/UX module in the form of categories. Once received, this module must connect to the SQL database containing the information on all available PDF files and use the proper stored procedure which will retrieve all relevant PDF files.

The stored procedure will return the paths of the PDF files, which the server side module must process into a list data structure, and send to the UI module for the population of its viewing queue and the subsequent rendering of files in that queue.

**6.2.2 Constraints**

As there is a separate UI module, this server side module does not shoulder any responsibility regarding the rendering of PDF files, and does not have direct access to the UI's components. Likewise, it also does not have the ability to access the tables stored in the database directly; it must retrieve the pertinent data through connecting and querying.

Regarding assumptions made by this module, if there are no requested filters coming from the user at that point in time, it will send the UI module a list containing the all paths to PDF files contained in the database.

**6.2.3 Composition**

The composition of the server side consists of 3 components data models, a view model and controller. The data model classes reflect the database tables and their contents. This also consists of a data manager class which reflects the stored procedures created within our database.

The view model component holds the properties and methods which are to be used in conjunction with the view. The model consists of 3 methods GetFilePath(), GetCategory(), and GetFilePathByCategory(). These 3 methods work together with the stored procedures created in our database. The GetFilePath() creates a list of all file paths in the database to send to the UI module. GetCategory() retrieves and creates a list of all category names in the database. It uses this list to populate a drop-down list in the UI module. GetFilePathByCategory() creates a unique list of file paths with the selection of one or many categories and sends this list to the UI module.

The controller component simply allows for logic flow from requests by the user and the corresponding methods in the view model.

**6.2.4 Uses/Interactions**

As aforementioned, this server side entity is the intermediary module between the UI/UX component and the database component of this product. It does not interact directly with the user, however, it is crucial to the process of filtering the PDF files as directed by the user, for the UI module is incapable of doing so on its own. The method of interaction with the UI comes down to sharing the proper parameters and variables with each other, while interaction with the database simply involves connecting to it and running the intended query.

**6.2.5 Resources**

All things considered, this server side module will not require a large amount of storage or resources relative to the UI/UX module or the database module, as it does not need to store large amounts of data internally, nor any graphical components as the UI will require.

However, the server side module will require memory to parse the user's selected categories received from the UI into a query which can be run through the database. It then must also have memory to store the paths received from said query into a list which will be stored in shared memory with the UI module.

**6.2.6 Interface/Exports**

Information regarding the specifics of implementation of the services provided by the server side module are, at the time of writing, incomplete, as the product is currently in the early stages of development. In the future, this document will be updated with the proper technical information for this section.

**6.3 PDF Database (Module)**

**6.3.1 Responsibilities**

This module is a SQL-based database which does not handle any sort of logic outside of running queries, and will solely be responsible for the storage of information, including the metadata and paths, of the PDF files. The server side module will connect to this database, and when necessary, will run queries on it. The database must return the proper information, namely the paths, to the relevant files, and send it back to the server side for processing.

**6.3.2 Constraints**

This module is far abstracted from the others, and does not have access to any code or memory in the first two modules as it is a database for storage only. Regarding assumptions made by this module, if there are no incoming 'filter' queries, the database will be queried by the server side module to return a list of every PDF file path available.

**6.3.3 Composition**

The composition of the database comprises 5 main tables which effectively sort the PDFs into pre-set categories laid out by a developer.

The first table, FileInfo, has 3 constituent columns; FileInfoID (primary key), FileName, and Size.

The second table, FileGroup, has 3 constituent columns; FileGroupID (primary key), Directory, and Description.

The third table, Categories, has 2 constituent columns; CategoryID (primary key) CategoryName.

The fourth table, FileGroup\_Files, has 2 constituent columns; FileGroupID (foreign key) and FileInfoID (foreign key). It relates the file info to a group it belongs to.

The fifth table, File\_Categories, has 2 constituent columns; FileInfoID (foreign key) and CategoryID (foreign key). It relates the file info to a category it belongs to.

**6.3.4 Uses/Interactions**

This database will only interact directly with the server side module. The user cannot type in their own queries or interact directly with the database in any capacity.

Information regarding the specifics of its interaction with the server side module are, at the time of writing, incomplete, as the product is currently in the early stages of development. In the future, this document will be updated with the proper technical information for this section.

**6.3.5 Resources**

The database will not necessarily be stored locally on the user’s device. However, the database will need a large amount of memory where it is stored to be able to maintain a large list of PDF files as the product client QTC requires.

**6.3.6 Interface/Exports**

Information regarding the specifics of implementation of the services provided by the database module are, at the time of writing, incomplete, as the product is currently in the early stages of development. The database schema may be slightly modified further into development. In the future, this document will be updated with the proper technical information for this section.

**6.4 Client Side (Module)**

**6.4.1 Responsibilities**

The responsibility of the client side module is to use the Atalasoft library to create a PDF viewer that the user may interact with. It is the responsibility of this module to use specific methods to open up and display PDFs within the viewer using only the physical path of the file. The module is responsible for retrieving such file paths from the aforementioned modules. Additionally, this module is responsible for handling any functionality that it is from direct use of the viewer. This module provides logic that allows the user to scroll from one PDF file onto the next. Additionally, this module contains the logic that allows a keyword search across all PDF files.

**6.4.2 Constraints**

As this is a client side module, it shoulders no responsibility regarding the querying and filtering of files. The module has no contact with the server as, thus it has no control over which file paths it receives. It simply takes the file paths it is given and makes sure to display them. Additionally, although the application is responsible for the creation of the viewer, it has no control over the viewer’s position and style within the webpage. This module simply requires an html element to be defined for specific use, then, the UI is responsible for placing that viewer in the correct position and changing its style.

**6.4.3 Composition**

The composition of this module consists of 3 major subcomponents. These subcomponents are client side javascript files.

The viewer-handler-script file is responsible for creating the PDF viewer and thumbnails at startup. Additionally, it is responsible for binding certain actions from the viewer/thumbnails to automatically trigger specific functions.

The page-change file is responsible for detecting changes in the viewer as the user is scrolling through the document, it contains additional logic that will aid the mechanism that allows the viewer to display different documents.. The PageNumber() function detects when the user has scrolled past a page within the viewer. It triggers the correct thumbnail so get highlighted as the user is scrolling onto a new page. This method detects the page number that the user has scrolled to and will initiate a document change if the user is trying to scroll past a document. The DocumentChange() method will trigger every time a new document gets loaded onto the viewer, it is responsible for adding pages at the beginning and at the end of the document that lets the user know they have reached the end/beginning of a document. The OnClickNext() and OnClickPrevious() will execute a document change when the corresponding “Previous / Next” buttons have been selected in the front end. These methods trigger the DocumentChange() as this method will trigger automatically on documents changing.

The search-function file is responsible for searching a keyword given by the user across all files that are currently in the list of files to be displayed. The OnClickSearch() initializes the search, it retrieves the keyword from the UI component and searches across multiple documents. The search will create a sub list that will prevent the user from seeing the files that do not contain the keyword. The user is only restricted when using the “Find Next / Previous” functions. SearchAll(), SearchAllHelper(), SearchAllSearch(), SearchAllCallback(), are all functions that work in conjunction with one another and execute the initial search and create the sublist of files that have been searched. The SearchAllHelper() initiates another search on the document that is currently being displayed in the viewer for the user, the SearchMatch() method scrolls the viewer to show the page where an instance of the keyword is and it highlights that keyword. The PreviousLoadSearch() and NextLoadSearch() handle the document change when the user decides to do an “onFindNext/ onFindPrevious” respectively. These load methods achieve the same task as the page-change methods to load documents, however, we kept them separate as we did not want to overlap these two subcomponents.

More details regarding these components are explained in the next section.

**6.4.4 Uses/Interactions**

These subcomponents and overall modules work with the UI as they will directly control the viewer object that is currently being displayed in the UI. The module has direct access to Atalasoft SDK library which includes methods and objects to be used for the direct creation and manipulation of the PDF viewer.

**6.4.5 Resources**

The modules do not store nor collect data outside of strings and objects within the code, thus, no allocation of memory is required for the client side module.

**6.4.6 Interface/Exports**

The set of services that are provided by this module are methods that are directly tied to UI components. Such methods are the following:

**7. Detailed Lower level Component Design**

**7.1 Details.cshtml**

**7.1.1 Classification**

Here various JavaScript and CSS scripts are called from the Atalasoft library for the pdf document to be able to be displayed on the web browser with the proper CSS design. Ultimately a function is implemented to load the pdf file at the start of the application, additional features as the tool bar are called in this function.

**7.1.2 Processing Narrative (PSPEC)**

**7.1.3 Interface Description**

HTML5 Web and Mobile VIewing: DotImage supports many formats including TIFF, PDF, DICOM, JPEG2000, JBIG2, Word, Excel, & PowerPoint. You can edit, insert, reorder, remove & rotate pages as well as cleanup documents using binarize, deskew & despeckle. DotImage includes Touch Support & Adaptive Scaling for Mobile Viewing and you can upload files using drag & drop or selection. A Thumbnail viewer is included to easily view and rearrange pages.

**7.1.4 Processing Detail**

**7.1.4.1 Design Class Hierarchy**

Parent class Atalasoft.Controls.WebDocumentsViewer ()

**7.1.4.2 Restrictions/Limitations**

DotImage interface allows us to display the PDF files onto the web browser however the purpose of our project is to display multiple PDF files which this interface does not allow.

**7.1.4.3 Performance Issues**

**7.1.4.4 Design Constraints**

The interface utilized in this Index.cshtml offers a basic design for the viewer, eventually our team will work on improving the CSS design of this to make it more user friendly.

**7.1.4.5 Processing Detail For Each Operation**

TBA currently the development part of this file will be changed and new thing we’ll be implemented

**7.2 page-change.js**

**7.2.1 Classification**

Here we have an ASP.NET handler file that connects/references DocViewerHandler(). It is in charge of changing documents seamlessly, and getting the document information when a new document is loaded.

**7.2.2 Processing Narrative (PSPEC)**

**7.2.3 Interface Description**

The OpenUrl method accepts a path url to the given document that will be displayed in the viewer.

The Next and Previous button switches to the next document available. Since the paths of the files are in a list, it simply calls the openurl method to open the document.

**7.2.4 Processing Detail**

The JavaScript file is client side meaning that it will run on the browser as its logic works through user input.

**7.2.4.1 Design Class Hierarchy**

This file has no Parent or Child classes. It is simply referenced in the view of the application.

**7.2.4.2 Restrictions/Limitations**

It has to be a pdf file and there are no limitations as it can open any file size.

**7.2.4.3 Performance Issues**

The testing for the logic in this file did not cause any performance issues.

**7.2.4.4 Design Constraints**

Scrolling and switching documents when reaching the end was a design constraint. Atalasoft did not have a way to determine when a client reached to end so we had to come up with our own way to determine it.

**7.2.4.5 Processing Detail For Each Operation**

**7.3 search-function.js**

**7.3.1 Classification**

Here we a client side JavaScript file that is in charge of providing the logic for the search mechanism that searches a keyword across multiple PDF files

**7.3.2 Processing Narrative (PSPEC)**

**7.3.3 Interface Description**

The OnClickSearch() method essentially fires the initial search. This method is called within the view, when the user clicks the “Search” button.

The OnClickNext() and OnClickPrevious() are public methods that will fire up the private method OnClickNextPrevious() which will essentially fire up the Atalasoft library to provide the next or previous search result to the user. These methods only work when the user has initiated a search through the OnClickSearch() method via the “Search” button.

The OnClickClear() method essentially resets private variables used to keep track of the search and it calls the Atalasoft library to search an empty string. Given that an empty string signifies to the Atalasoft library that the user would like to clear the search. This method gets fired through the index via the “Clear” button

**7.3.4 Processing Detail**

The JavaScript file is client side meaning that it will run on the browser as its logic works through user input.

**7.3.4.1 Design Class Hierarchy**

This file has no Parent or Child classes. It is simply referenced in the view of the application.

**7.3.4.2 Restrictions/Limitations**

The file must be referenced in a view as a script in order to provide its interface. Similarly, all references the file makes to objects, data structures, and methods outside of the scope of the file must be present in another JavaScript file that has also been referenced in a view.

**7.3.4.3 Performance Issues**

The testing for the logic in this file did not cause any performance issues.

**7.3.4.4 Design Constraints**

The “find previous keyword” logic within this file has a design constraint. The requirement asks us that if the user is currently looking at the first instance of a keyword (first page the word is found) and they click “Find Previous” the program should display the previous document and show the last instance of the keyword within that document (last page the word is found). However, Atalasoft does not provide a way to grab the last instance of a keyword as it searches documents from top to bottom, thus it always gives us the first instance to work with. In order to show the last instance, we then do a small run through the document until we reach the last instance. This takes a more time for the user to see the previous document and keyword when they are going backwards as opposed to if the user was just simply going forward.

**7.3.4.5 Processing Detail For Each Operation**

**7.4 PDFViewerController.cs**

**7.4.1 Classification**

Here this class serves as the logic flow between the user requests and the view model and it’s methods.

**7.4.2 Processing Narrative (PSPEC)**

**7.4.3 Interface Description**

Handles requests from the view and updates the view model via a do HttpPost method.

**7.4.4 Processing Detail**

**7.4.4.1 Design Class Hierarchy**

This class has no parent or child classes.

**7.4.4.2 Restrictions/Limitations**

If the database and stored procedures are not set up correctly it will not allow the application to continue running as the flow in logic will be incomplete.

**7.4.4.3 Performance Issues**

The testing for the logic in this file did not cause any performance issues.

**7.4.4.4 Design Constraints**

Designed to catch errors when processing requests from the user side.

**7.4.4.5 Processing Detail For Each Operation**

**7.5 PDFViewerViewModel.cs**

**7.5.1 Classification**

This class serves as the model for our view and houses the properties and methods to be used in the view.

**7.5.2 Processing Narrative (PSPEC)**

**7.5.3 Interface Description**

The GetCategory() method uses a stored procedure to generate a list of all categories in the database. These categories will be used in a drop-down-box in the view. The GetFilePath() method uses a stored procedure to create a list of all file paths stored in the database to send to the viewer. The GetFilePathByCategory() method also utilizes a stored procedure to create a list of unique file paths based on the category or categories chosen by the user. The load() method is used to set the initial values in the view such as the categories and the list of file paths to load in the viewer.

**7.5.4 Processing Detail**

**7.5.4.1 Design Class Hierarchy**

This class has no parent or child classes.

**7.5.4.2 Restrictions/Limitations**

The methods used in this class utilize the stored procedures created in the database. Thus if the database nor the stored procedures are created neither the methods nor the properties will operate successfully.

**7.5.4.3 Performance Issues**

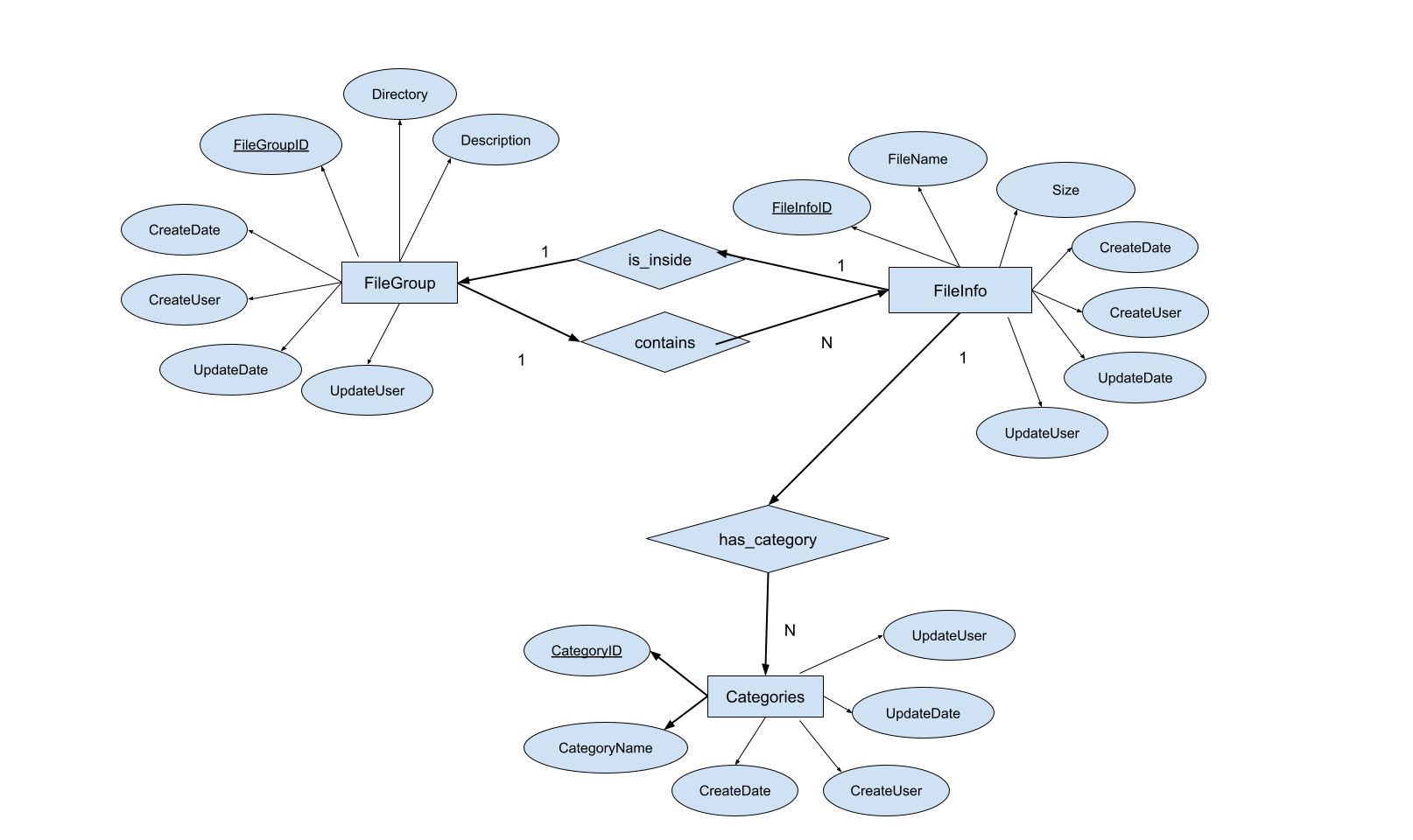
The testing for the logic in this file did not cause any performance issues.

**7.5.4.4 Design Constraints**

Designed to easily add more functionality or properties if needed.

**7.5.4.5 Processing Detail For Each Operation**

**8. Database Design**



The database will consist of five tables. The first table will be “FileGroup” which holds data pertaining to the folders of PDFs on the system. It will consist of 3 columns. The first column “FileGroupID” will be a unique ID and the primary key for the table. The second column would be the “Directory” which will hold reference to the location of the folder on the system. The last column “Description” will have a description of what is in the folder.

The second table “FileInfo” will hold information pertaining to the individual files. It will consist of three columns. The first column will be “FileInfoID” which is a unique primary key. The second column is “FileName” which will be a string of the file’s name on the system. The last column will be “Size” which will be the size of the file on the system.

The third table “FileGroup\_Files” will be a table referencing the files and each folder they pertain to. It will consist of two columns. The first column will be “FileGroupID” which will be a foreign key referenced from table “FileGroup”. The second column will be “FileInfoID” which will be a foreign key referenced from table “FileInfo”.

The fourth table “Categories” will be a table holding the possible categories for each PDF. It will consist of two columns. The first column “CategoryID” will be a unique primary key. The second column will be “CategoryName” which will hold a string of possible classifiers.

The fifth table “File\_Categories” will hold reference to each PDF and what categories they can be classified as. It will consist of two columns. The first column “FileInfoID” will be a foreign key referenced from table “FileInfo”. The second column will be “CategoryID” which will be a foreign key referenced from table “Categories”.

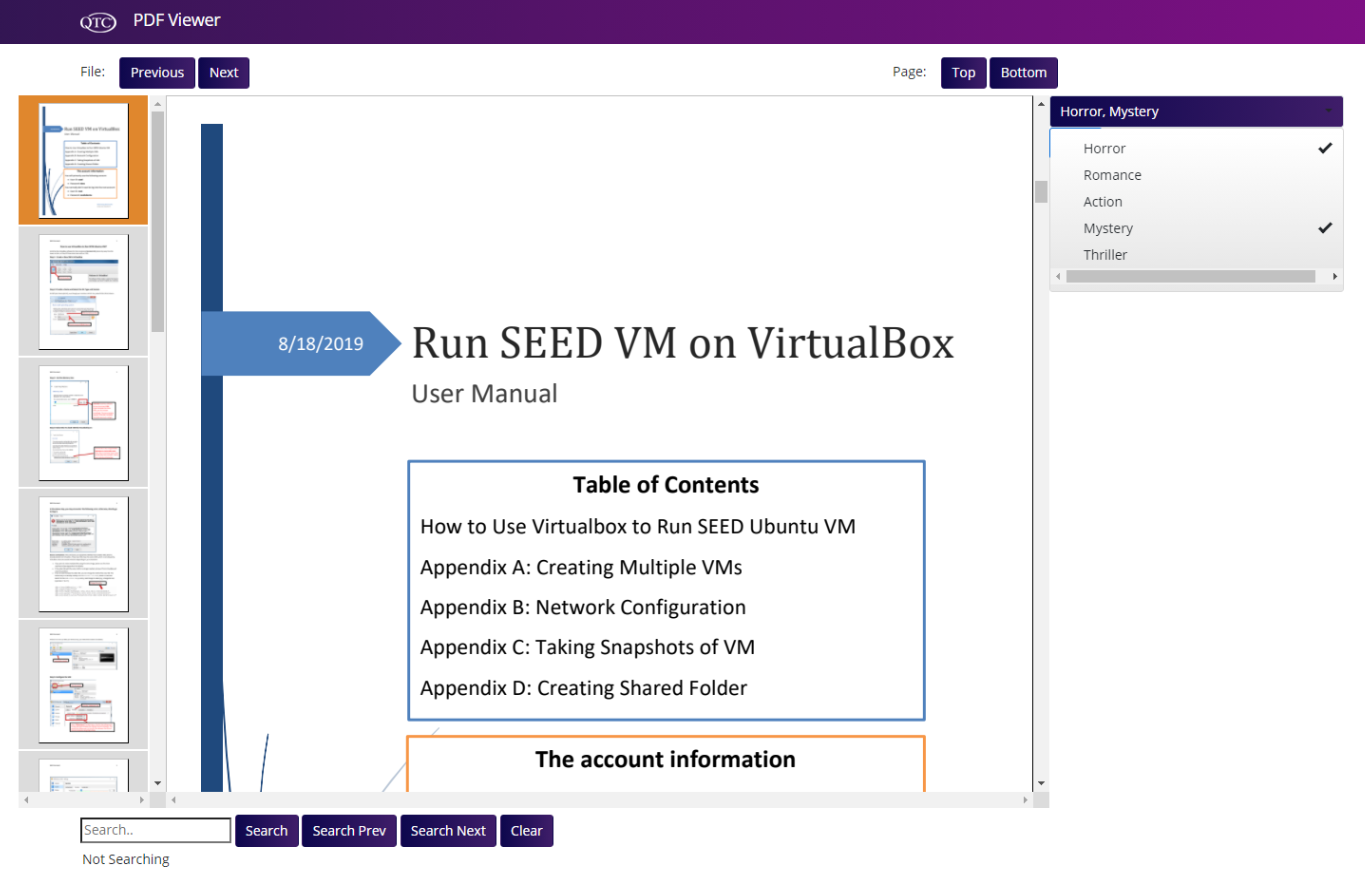
All tables will consist of columns “CreateDate”, “CreateUser”, “UpdateDate”, and “UpdateUser”. These columns are implemented for QTC to keep track of who and when the database is updated.

**9. User Interface**

**9.1 Overview of User Interface**

The User shall be able to open PDFs into a viewer. Once the PDFs are open into view the user will be able to scroll through the many pdfs by scrolling down. At the top left of the viewer there will be 2 buttons to instantly go to the previous or next PDF in the list. At the top right there will also be 2 buttons to scroll instantly to the top or bottom of the PDF in view. Along the left side of the viewer there will be a thumbnailer showing every page in the current PDF document. The thumbnailer allows the user to click and instantly go to the selected page. To the right of the viewer there will be a drop-down list which will allow the user to filter through the categories mentioned in the list. This will update the PDFs shown in the viewer to correspond to the related categories. Below the viewer there will be a search bar to search for keywords in the text of the PDFs. Along with a Search Next and Search Previous button which will allow the user to go between each instance of the matching keyword.

**9.2 Screen Frameworks or Images**



**9.3 User Interface Flow Model**

This section will be completed once we have a working prototype of the product. This should include a flow diagram of the navigation between different pages, as well as a discussion of screen objects and actions associated with those objects.

**10. Requirements Validation and Verification**

|  |  |  |  |
| --- | --- | --- | --- |
| Requirement No | Requirement Description | Component Module | Testing Requirement |
| 1.1.1 | The system shall implement an MVC ASP.NET web solution. |  |  |
| 1.1.2 | The system shall place multiple pdf files in a queue to be displayed. |  |  |
| 1.1.3 | The system shall let the user scroll onto the next file in the queue after viewing the current pdf file. |  |  |
| 1.1.4 | The system shall allow the user to search for text specific keywords within the pdfs content. |  |  |
| 1.1.5 | The system's search feature shall search through all of the pdfs that are in the display queue. |  |  |
| 1.1.6 | The system's search feature shall allow the user to navigate back and forth through each occurrence of the searched text. |  |  |
| 1.1.7 | The system shall update the display queue based off on the categories selected by the user. |  |  |
| 1.1.8 | The system shall connect to an SQL database. |  |  |
| 1.1.9 | The system shall make an SQL query call to retrieve the list of all categories. |  |  |
| 1.1.10 | The system shall make an SQL query call to retrieve the PDF file locations of PDFs that must be displayed. |  |  |
| 1.2.1 | The interface shall be an online PDF viewer. |  |  |
| 1.2.2 | The interface shall integrate Atalasoft's DotImage web viewer to display PDF files. |  |  |
| 1.2.3 | The interface shall use Atalasoft’s built-in toolbar. |  |  |
| 1.2.4 | The interface shall display multiple pdf files in a single stream as if dealing with one large pdf file. |  |  |
| 1.2.5 | The interface shall have a filter by category feature that will allow the user to select specific categories. |  |  |
| 1.2.6 | The interface shall prompt the user with a list of categories to choose from. |  |  |
| 1.2.7 | The interface shall allow the user to select multiple categories. |  |  |
| 1.3.1 | The database shall store the file locations of all PDFs within a table. |  |  |
| 1.3.2 | The program's database shall store the file names of all PDFs within a table. |  |  |
| 1.3.3 | The database shall store all possible categories within a table. |  |  |
| 1.3.4 | The database shall have a table that displays the relation between a file and its category. |  |  |
| 1.3.5 | The database shall have a table that displays the relationship between file names and its file group. |  |  |

**11. Glossary**

This Section will represent an ordered list of defined terms and concepts that are listed through the document. We need to provide definitions for relevant terms, abbreviations, and acronyms. The list, at the time of writing, cannot be defined as complete due to the fact that in the future, new terms may be added to the document which may also be added to this list. For now, some definitions that will be displayed here are for words used in this paper, such as: SDK, PDF, CSS, HTML, SQL, MVC, and queue.

**12. References**

Atalasoft DotImage documentation: <https://www.atalasoft.com/docs/dotimage/docs-concept/AtalasoftDotImageDevelopersGuide_EN.pdf>

SQL Server documentation: <https://docs.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver15>